System Architecture
The relationship between the kernel and the system hardware environment

Michael E. Locasto

Department of Computer Science
UofC
CPSC 457

September 22, 2014
Agenda

News (1 minute)
→ homeworks will be released soon
→ USB3.0 stick available Tuesday

Learning Objectives (1 minute)
Recall “What is OS?” Discussion (1 minute)
→ revisit “Big Picture”

Overview / Refresh of x86 (15 minutes)
→ gdb of silly programs

IA-32 system architecture (25 minutes)
→ system states, privilege rings (CPU modes)
→ key control registers
→ GDT, IDT
The last set of slides ended with 3 questions:

How does a kernel enforce these divisions?
What is the necessary architectural support for basic operating system functionality?
How do processes ask the operating system to do stuff?

Suspend discussion thread of “how does a program become a process?” at “the fork/execve system calls” fwd pointer to system calls
Learning Objective

By reviewing the IA-32 architecture, students should be able to distinguish between the *execution environment* for userland processes and the *programmable resources* available to the system supervisor (i.e., the kernel).

Students should understand that when the CPU is in *protected mode*, the privileged control infrastructure and instructions are available to a piece of system software (i.e., a “supervisor”) running in *ring 0*. Students should also understand the role that *interrupts* and *exceptions* (managed via the IDT) play in managing *context switching* between *userland* and *kernel space*, and they should understand the role that the GDT and segment selectors play in supporting isolation between the kernel and user space.
Figure 2-3. Transitions Among the Processor’s Operating Modes

The processor is placed in real-address mode following power-up or a reset. The PE flag in control register CR0 then controls whether the processor is operating in real-address or protected mode. See also: Section 9.9, “Mode Switching,” and Section 4.1.2, “Paging-Mode Enabling.”
Note that, as the textbook says, OSs started out by replacing the human operator: which is one reason we emphasize part of the definition of an OS as something that loads and prepares software for execution.

Main issue: how can the OS simultaneously time-multiplex the CPU(s) for all those processes? But there is another aspect, since sharing the CPU does not imply protection / isolation: space multiplexing memory and the isolation / protection implied.
Q: What separates user and kernel mode?
A: CPL bits in CPU, DPL bits in segment descriptors

Q: What separates process address spaces?
A: We’ll cover that in a couple of sessions.
goto gdb
IA-32 Supervisor Environment

Programmable Resources

Figure 2-1. IA-32 System-Level Registers and Data Structures
Figure 6-1. Relationship of the IDTR and IDT
Interrupts

The CPU can service interrupts from both hardware devices and software. Pure interrupts often come from devices that need to tell the kernel they have data to deliver to a process.

Software–initiated interrupts are often called exceptions; exceptions can happen because of programming errors (e.g., divide-by-zero) or important conditions (such as executing an x86 int3 instruction). This second type of exception is often called a “software interrupt.”

Confused enough?
The IDT is a table located in memory (address is in the IDTR). It contains 256 entries of 8 bytes each (2048 bytes).
These interrupts can be invoked from the userland (ring3).

```c
set_system_gate(3,&int3)
set_system_gate(4,&overflow)
set_system_gate(5,&bounds)
set_system_gate(0x80,&system_call);
```

In `/linux/arch/i386/kernel/traps.c::set_trap_gate(n, addr)`
insert a trap gate with the DPL field set to 0.
The Others exception are initialized with `set_trap_gate`:

```c
set_trap_gate(0,&divide_error)
set_trap_gate(1,&debug)
set_trap_gate(2,&nmi)
set_trap_gate(6,&invalid_op)
set_trap_gate(7,&device_not_available)
set_trap_gate(8,&double_fault)
set_trap_gate(9,&coprocessor_segment_overrun)
set_trap_gate(10,&invalid_TSS)
set_trap_gate(11,&segment_not_present)
set_trap_gate(12,&stack_segment)
set_trap_gate(13,&general_protection)
set_trap_gate(14,&page_fault)
set_trap_gate(15,&spurious_interrupt_bug)
set_trap_gate(16,&coprocessor_error)
set_trap_gate(17,&alignment_check)
set_trap_gate(18,&machine_check)
```
Segment Descriptors

GDT

Figure 5-1. Descriptor Fields Used for Protection
Privilege Rings

Figure 5-3. Protection Rings
Supplemental Reading
This is background reference material

The Intel IA32 Developer Manuals, Volume 1
http://www.intel.com/products/processor/manuals/
(a) Section 2.1 (skim this)
(b) Section 2.2 (paying particular attention to the description of the pipelined microarchitecture)
(c) Chapter 3 (paying particular attention to 3.1 (and Figure 3-1), 3.2, 3.3, 3.4)

The Intel IA32 Developer Manuals, Volume 3A:
http://www.intel.com/products/processor/manuals/
(a) Chapter 2 (paying special attention to:)
(b) Figure 2-1
(c) Section 2.2
(d) Figure 2-3
(e) Section 9.1